Marja-Leena Leppäaho

3D-peliympäristön Toteuttamisen Työjärjestys ja Optimointi

Metropolia Ammattikorkeakoulu

Medianomi (AMK)

Viestintä

Opinnäytetyö

10.11.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Marja-Leena Leppäaho 3D-peliympäristön Toteuttamisen Työjärjestys ja Optimointi 34 sivua + 2 liitettä 10.11.2017
Tutkinto	Medianomi (AMK)
Koulutusohjelma	Viestintä
Suuntautumisvaihtoehto	3D-animointi ja -visualisointi
Ohjaaja(t)	Lehtori Kristian Simolin

Tässä opinnäytetyössä tutkitaan ja käytännössä sovelletaan 3D-peliympäristöjen grafiikan optimointia ja peliympäristöjen luomiseen sopivaa työjärjestystä. Työn tavoitteena on esitellä lukijalle toimiva työjärjestys hyvin optimoitujen peliympäristöjen luomiseen. Opinnäytetyö koostuu teorian- ja käytännönosiosta. Luvun 2 teoriaosuudessa tutkitaan erilaisia optimointikeinoja kirjallisuuden ja muiden lähteiden pohjalta. Luvun 3 käytännön osiossa optimointi yhdistetään peliympäristöjen luomisessa käytettyyn työjärjestykseen.

Teoriaosuudessa käydään yleisellä tasolla läpi erilaisia optimointikeinoja ja suorituskyvyn mittauksen perusteita PC-peliympäristöjä luotaessa. Käsiteltäviin optimointikeinoihin kuuluvat mallien ja tekstuurien optimointi, näkyvyysoptimointi, niputtaminen, instansointi sekä valaistuksen optimointi. Tutkimisen jatkaminen on kuitenkin tärkeää optimointimenetelmien käytön jatkuvan muuttumisen ja nopean kehittymisen vuoksi. Tässä opinnäytetyössä ei ole myöskään lueteltu kaikkia mahdollisia optimointimenetelmiä, vaan keskitytty PC-pelien kehityksessä käytettyihin yleisimpiin tekniikoihin.

Käytännön osiossa seurataan alusta loppuun 3D PC-peliympäristön rakentumista erilaisten työvaiheiden kautta ja sitä, miten optimointi toteutuu eri työvaiheissa. Työjärjestys koostuu konseptoinnista, blokkauksesta, mallintamisesta, teksturoinnista, valaistuksesta ja kameran jälkiefektien käyttämisestä. Esitelty työjärjestys ei ole ainoa mahdollinen, vaan se saattaa vaihdella paljonkin esimerkiksi pelin laajuudesta ja tiimin koosta riippuen. Esitetty järjestys antaa kuitenkin hyvän kuvan tyypillisistä työvaiheista ympäristöjä luotaessa.

Hyvä optimointi ja työjärjestys auttavat graafikkoja luomaan monipuolisia ympäristöjä sekä vähentämään ongelmia ja hidasteita projektin aikana. Opinnäytetyön kautta on tavoitteena saada hyvä kokonaiskuva käytössä olevista optimointikeinoista ja niiden käytöstä peliympäristöjä luotaessa sekä toimia oppaana aihepiiristä kiinnostuneille 3D-graafikoille.

Avainsanat	3D, reaaliaikainen, peliympäristö, optimointi, työjärjestys

Metropolia

Author(s) Title Number of Pages Date	Marja-Leena Leppäaho Workflow and optimizations for creating 3D game environments 34 pages + 2 appendices 10 November 2017
Degree	Bachelor of Culture and Arts
Degree Programme	Media
Specialisation option	3D Animation and Visualization
Instructor(s)	Kristian Simolin, Senior Lecturer

This thesis studies and tests the graphic optimizations and the creation workflow of 3D game environments. The thesis aims to present the reader with a good basic workflow for creating optimized game environments. The thesis is divided into two parts: theory and practice. The theory part in chapter 2 consist of collecting the research material on optimization methods. In the practice part in chapter 3, the optimization methods are combined with the game environment creation workflow.

In the theory section of the thesis, different kinds of optimisation methods and basics of performance measuring when creating PC game environments are covered. The optimization methods discussed in this chapter include the optimisation of models and textures, visibility culling, LODs, mipmaps, batching, instancing and the optimisation of lighting. The use and development of optimisation methods is in constant change so it is important to continue the research. All known optimisation methods could not be included in this thesis so the focus is on the most common techniques used in the development of PC-games.

In the practical section of the thesis, the creation and optimisation of a game environment is followed from the start to the finish. The workflow consists of concepting, blocking, modelling, texturing, lighting and the choosing of post effects. The presented workflow is not exclusive, workflows can vary depending on the scale of the game or development team size. However, the workflow presented here gives a good outline of the typical stages in game environment creation.

Good optimisation and workflow helps graphic artists to create diverse game environments and reduce problems and roadblocks during projects. The aim of this thesis is to create a comprehensive list of commonly used optimization methods and their use in the process of creating game environments as well as to work as a guide for 3D-graphics artist interested in this field.

Keywords	3D, real-time, game environment, optimization, workflow

Metropolia

Sisällys

1 Johdanto

Tässä opinnäytetyössä esitellään esimerkkiprojektin kautta yksi mahdollinen työjärjestys peliympäristön toteuttamiselle ja käydään läpi yleisimpiä metodeja pelien ja erityisesti peliympäristöjen 3D-grafiikan optimointiin. Esimerkkiprojektina toteutetaan reaaliaikaisesti pyörivä realistinen PC-peleille optimoitu ympäristö.

Opinnäytetyössä ei toteuteta tietokonepeliä tai pelin muita toiminnallisuuksia, vaan syvennytään PC-peleille suunnatun 3D-peligrafiikan ja -peliympäristöjen optimointiin. Työssä käydään läpi peliympäristön rakentamisen vaiheet ja optimointi konseptista blokkaukseen, mallintamiseen, teksturointiin, valaistukseen ja kameran jälkiefekteihin. Työssä ei oteta kantaa esimerkiksi pelin tai pelikenttien suunnitteluun tai pelaajan ympäristössä ohjailuun. Opinnäytetyössä esiteltävä työjärjestys ei ole ainoa mahdollinen, vaan se saattaa vaihdella muun muassa laajuudeltaan paljon eri pelistudioissa aina pelin ja peliympäristön laajuudesta ja tiimin koosta riippuen. Esitetty järjestys antaa kuitenkin hyvän kuvan tyypillisistä työvaiheista ympäristöjä luotaessa.

Opinnäytetyö on suunnattu pelialasta ja peliympäristöistä kiinnostuneille 3D-graafikoille. Oletuksena opinnäytetyössä on lukijan ymmärrys mallintamisen, teksturoinnin ja pelimoottoreiden perusteista. Esimerkeissä käytetään Unreal Engine-, Autodesk Maya- ja Substance Painter -ohjelmistoja, joissa syntyvät tulokset voidaan tuottaa myös muilla vastaavilla tekniikoilla, minkä vuoksi näiden ohjelmistojen käyttöön ei syvennytä tässä opinnäytetyössä sen tarkemmin.

Luvussa 3 käydään yleisellä tasolla läpi optimoinnin eri keinoja, joita 3D-graafikon tulisi ottaa huomioon suunnitellessaan ja toteuttaessaan 3D-peliympäristöä PC-laitteille. Luvussa esitellään ensin, mihin suorituskyky perustuu ja miten sitä mitataan, minkä jälkeen käydään läpi optimointiin liittyviä tekniikoita, kuten mallinnuksen ja tekstuurien optimointi, näkyvyysoptimointi, niputtaminen, instansointi ja valaistuksen optimointi.

Luvussa 4 seurataan esimerkkiympäristön luomista esitetyssä työjärjestyksessä ja toteutetaan käytännössä luvun 3 optimointikeinoja. Työjärjestys koostuu konseptoinnista, blokkauksesta, mallintamisesta, teksturoinnista, valaistuksesta ja kameran jälkiefektien käyttämisestä.

2 Käsitteiden määrittely

Peliympäristö – Pelimoottorissa luotu reaaliaikaisesti pyörivä ympäristö, jossa pelaaja liikkuu.

Optimointi – Parhaimman vaihtoehdon, tilan, määrän tai ihanneolosuhteiden hakeminen.

Työjärjestys – (engl. *workflow*) toistettava järjestys tai malli ketjulle, jossa erilaisia toimintoja tehdään.

Pelimoottori – Pelien luomiseen ja kehittämiseen tarkoitettu ohjelmistokehys. Tunnettuja pelimoottoreita ovat esimerkiksi Unreal Engine, Unity, CryEngine ja GameMaker.

Kuvataajuus – Näytölle piirrettyjen kuvien määrä aikayksikköä kohden. Mittayksikkönä usein FPS (*frames per second*).

Tahko – (engl. *face*) Myös polygoni. Vähintään kolmesta särmästä koostuva monikulmio 3D-mallissa.

Verteksi – 3D-mallin kärkipisteet.

Särmä – (engl. *edge*) 3D-mallin tahkojen vierekkäisten reunaverteksien välille piirtyvät viivat.

Polygonimäärä – 3D-mallin resoluutio eli tahkojen määrä.

Topologia – Verteksien ja särmien asettelu 3D-mallin pinnan luomiseksi.

Staattinen objekti – peliobjekti, kuten esimerkiksi 3D-malli tai valo, joka ei muutu tai liiku pelin aikana.

Dynaaminen objekti – peliobjekti, kuten esimerkiksi 3D-malli tai valo, joka muuttuu tai liikkuu pelin aikana.

3 PC-peliympäristöjen grafiikan optimointi

Tietokonepelit edustavat reaaliaikaista grafiikkaa. Jokainen kuva täytyy renderöidä eli tuottaa näytölle juuri silloin, kun sitä tarvitaan. Kuvan sulavan liikkeen ja pelaajalle miellyttävän kokemuksen saavuttamiseksi tarvitaan pelien optimointia niin ohjelmoijilta, suunnittelijoilta kuin graafikoilta.

Tässä luvussa keskitytään erilaisiin tekniikoihin, joita graafikot voivat käyttää hyvän suorituskyvyn aikaansaamiseksi. Luvussa käydään ensin pohjustavana teoriana yleisellä tasolla läpi FPS:n merkitystä ja laskennan jakautumista CPU:n ja GPU:n välille. Sen jälkeen luvussa esitellään mallinnukseen, tekstuureihin ja materiaaleihin liittyviä optimointitekniikoita, näkyvyysoptimointia, niputtamista, instansointia ja valaistuksen optimointia. Optimointi voi tapahtua automaattisesti pelimoottorin puolesta, pelimoottorin tarjoamilla työkaluilla tai vaatia graafikon huomiota projektin alusta lähtien. Kaikista osa-alueista on kuitenkin hyvä olla tietoinen ympäristöjä luotaessa.

3.1 FPS, CPU ja GPU

Tärkein mittari pelien ja peliympäristöjen hyvälle suorituskyvylle ja täten onnistuneelle optimoinnille on pelin kuvataajuus eli FPS (engl. *Frames Per Second*). FPS kertoo, kuinka monta kuvaa sekunnissa tietokone tuottaa näytölle.

PC-pelien kehityksessä tähdätään yleensä 30–60 FPS:n välille. Tätä korkeammat FPS:t ovat suotuisia, mutta usein vaikeampia saavuttaa. Alhaisemmat FPS:t taas alkavat jo merkittävästi heikentämään pelaajan kokemusta kuvan pätkittäisen liikkeen vuoksi. Myös pelin genrellä voi olla väliä, sillä esimerkiksi nopeita reaktioita vaativat ensimmäisen persoonan ampumispelit tarvitsevat yleensä vähintään 60 FPS:n jatkuvan kuvataajuuden. (Klappenbach 2017.)

Kuvataajuudesta vastaavat erityisesti tietokoneen CPU (engl. *Central Processing Unit*) eli suoritin ja GPU (engl. *Graphics Processing Unit*) eli grafiikkasuoritin. Laskennan vähentäminen ja jakaminen tasaisesti näiden järjestelmien välille parantavat

ympäristön kokonaissuorituskykyä, koska kuvataajuutta rajoittaa näistä aina hitaammin toimiva komponentti. (O'Conor 2017.)

CPU käy läpi ympäristössä olevat 3D-mallit yksitellen ja laskee niihin vaikuttavat asiat, kuten esimerkiksi valot, tekstuurit ja materiaalit. Kerättyään tarvittavat tiedot 3D-mallista CPU lähettää ne GPU:lle. Tätä CPU:lta GPU:lle lähetettävää tietopakettia kutsutaan piirtokutsuksi, eli toisin sanottuna CPU kutsuu GPU:n piirtämään eli renderöimään mallin antamiensa ohjeiden mukaan. (Gesota 2016b.) Piirtokutsuja lähetetään yleensä vähintään yksi ympäristöstä renderöitävää 3D-mallia kohden, mutta esimerkiksi useiden materiaalien ja läpinäkyvyyden käyttö samassa 3D-mallissa lisäävät usein piirtokutsujen määrää (O'Conor 2017).

Yksi piirtokutsu vie pienen määrän aikaa CPU:lta, mutta kasautuessaan piirtokutsut voivat ruuhkauttaa CPU:n toimintaa. Esimerkiksi 3D-ympäristöön luotu muuri, joka on tehty 10 000 erillisestä tiilestä, aiheuttaa CPU:lle vähintään 10 000 piirtokutsua. Jos koko tiilimuuri yhdistettäisiin yhdeksi 3D-malliksi, olisi se CPU:lle 10 000 kertaa nopeampi prosessoida. Myös yhdistelemällä ja vähentämällä malleihin vaikuttavia asioita, kuten esimerkiksi valoja ja tekstuureita, voidaan piirtokutsuja vähentää. (Cober 2017.)

GPU käyttää CPU:n piirtokutsusta saamiansa tietoja ja tuottaa niiden avulla lopullisen kuvan näytölle. Toisin kuin CPU:n, GPU:n työtä hidastavat 3D-mallien suuret verteksimäärät ja tekstuurikarttojen suuri koko. (O'Conor 2017.) Näitä manipuloimalla voidaan parantaa GPU:n suorituskykyä tai tarvittaessa jakaa kuormitusta CPU:n ja GPU:n välille.

Useimmat pelimoottorit tarjoavat erilaisia profilointityökaluja, joiden avulla voi selvittää, miten laskenta on jakautunut, miten FPS vaihtelee ympäristössä ja mitkä mallit tai materiaalit aiheuttavat ongelmia suorituskyvylle (O'Conor 2017). Työkaluja kannattaa käyttää aktiivisesti alusta alkaen, jotta ongelmakohdat tulisivat huomatuiksi mahdollisimman nopeasti.

3.2 Mallintaminen

Mallien polygonimäärien silmällä pitäminen voi tuntua monesta graafikosta jo vanhanaikaiselta nykyaikaisten tehokkaiden tietokoneiden kanssa työskenneltäessä.

Täytyy pitää mielessä, että kaikki hyvin optimoidut osa-alueet yhdessä mahdollistavat lopulta runsaamman lopputuloksen. Kun huolehtii resurssien käytöstä, niitä on myös paljon enemmän käytettävissä. Polygonien säästäväinen käyttö mallin siluettia hukkaamatta, turhien pintojen poistaminen paikoista, joissa ne eivät näy, ja pienempien yksityiskohtien lisääminen malliin normaalikarttojen avulla ovat kaikki menetelmiä, joita käyttämällä mallit on edelleen hyvä luoda.

Polygonimäärä on käsite, jota graafikot ovat tottuneet käyttämään aikojen saatossa kuvailemaan mallien raskautta. Perinteisten nelikulmioisten polygonien sijaan pelimoottorit käsittelevät 3D-malleja kolmioina, ja optimoinnin näkökulmasta polygonimäärää tärkeämpää olisi keskittyä mallien verteksimääriin (Polycount Wiki n.d.).

Kuvio 1. Saman mallin eri verteksimäärät mallinnusohjelmistossa ja Unreal Enginessä.

Yleensä mallinnettaessa kolmiomäärät ja verteksit ovat toisiinsa suoraan verrannollisia: yksi kolmio vastaa kolmea verteksiä. Se, miksi vertekseihin on peligrafiikassa kiinnitettävä huomiota, johtuu mallien UV-saumoista, skinnauksesta, verteksien värimäärittelyistä, materiaalivaihdoksista ja malleissa käytettävistä pehmeistä tai kovista särmistä (engl. *smoothing groups*). Ne vaikuttavat mallin renderöintiin, jonka vuoksi näihin kohtiin verteksien määrä voi laskennallisesti moninkertaistua GPU:lle siirryttäessä. (Unity Manual n.d.) Liian pieniin paloihin pilkotut UV:t, mallien riggaaminen, liian monet materiaalit samassa mallissa ja runsas vaihtelu pehmeiden ja

kovien särmien välillä kaikki vaikuttavat mallin raskauteen, minkä vuoksi niiden vaikutuksesta on hyvä olla tietoinen.

Kun 3D-mallit siirretään pelimoottoriin, niille tarvitsee usein asettaa myös näkymättömät törmäyspinnat (engl. collider). Törmäyspinnat havaitsevat muiden mallien läpitulon ja usein esimerkiksi estävät pelaajaa kävelemästä mallien lävitse. Usein törmäyspinnoiksi voidaan malleille asettaa erilaisia primitiivimuotoja, kuten kapseleita tai laatikoita. Monimutkaisten törmäyspintojen käyttö on suorituskyvylle raskasta, minkä vuoksi pelaajalle näkymättömissä törmäyspinnoissa tulisi suurimman osan ajasta suosia primitiivimuotoja tai alemman polygonimäärän versioita käytetystä mallista. (Gonzalez 2017.) Voi myös olla hyödyllistä jättää törmäyspinnat kokonaan laskematta, jos ne sijaitsevat tarpeeksi kaukana kamerasta (Ahearn 2008, 28.)

3.3 Tekstuurit ja materiaalit

Tekstuurien tekeminen alkaa jo mallinnusvaiheessa luotaessa UV-karttoja 3D-malleille. Tehokkaasti optimoidut UV-kartat vähentävät tarvittavien tekstuurien tiedostokokoa ja resoluutiota sekä nopeuttavat GPU:n toimintaa. UV:ita luotaessa tilaa voi säästää asettelemalla UV-saarekkeita päällekkäin, vaihtelemalla prioriteetin mukaan niiden kokoa kartan alueella tai käyttämällä toistuvia tile-tekstuureita (McKinley 2005, 100–114). Myös mahdollisimman vähien UV-saarekkeiden käyttö nopeuttaa GPU:n suorituskykyä (Polycount Wiki n.d.).

Staattisia malleja yhdistelemällä voidaan tehokkaasti vähentää CPU:n luomia piirtokutsuja, mutta jos yhdistellyt mallit käyttävät eri tekstuureja, suorituskyky ei parane. Tekstuuriatlaksen tarkoitus on yhdistää useampi tekstuuri samaan karttaan ja näin mahdollistaa mahdollisimman monenlaisten mallien yhdistäminen toisiinsa. Tekstuuriatlaksien luominen myös helpottaa projektin organisointia vähentämällä kansiorakenteessa olevien liikkuvien osasten määrää (Gonzalez 2017). Atlaksien luominen itse johtaa usein parhaisiin tuloksiin, mutta ajan säästämisen vuoksi niiden kokoamiseen käytetään usein pelimoottorien tarjoamia työkaluja. GPU:n laskennan optimoimiseksi kaikkia tekstuureita etenkään suuremmissa projekteissa ei ole hyödyllistä yhdistellä yhteen jättiläismäiseen karttaan, vaan atlaksiin kannattaa valikoida tekstuureita malleista, joita useimmin yhdistellään toisiinsa. Tekstuuriatlaksiin ei myöskään voi soveltaa toistuvia tile-tekstuureita. (Ivanov 2006.)

Tekstuurikartat tai -atlakset viedään pelimoottoriin yleensä JPG- tai PNG-muodoissa. GPU ei kuitenkaan kykene lukemaan näitä formaatteja suoraan renderöidessään 3D-grafiikkaa, vaan vaatii ensin niiden uudelleenpakkauksen. Useimmat pelimoottorit pakkaavat tekstuurikartat automaattisesti GPU:lle sopivaan muotoon. GPU:n lukemistyön mahdollistamisen lisäksi tekstuurikartan tiedostokoko voi pienentyä jopa neljännekseen ilman kuvan laadun merkittävää heikentymistä. (Gesota 2016a.)

Kuvio 2. Sama kuva PNG- ja DXT1-pakkauksilla.

Parhaan optimoinnin aikaansaamiseksi on tärkeää valita oikea pakkausmenetelmä oikealle tekstuurille. Kaikki kohdelaitteet eivät kuitenkaan tue kaikkia pelimoottorin tarjoamia pakkausmenetelmiä. Yleisesti ottaen PC-peleissä käytetään usein ASTC- tai DXT-pakkauksia, Android-mobiililaitteilla ASTC- ja ETC-pakkauksia ja IOS-laitteilla PVRTC-pakkauksia. (Gesota 2016a.) Kaikki pakkausmenetelmät eivät myöskään tue läpinäkyvyyttä eli alfakanavaa, esimerkiksi DXT5-pakkauksessa alfakanava on mukana mutta DXT1-pakkauksessa ei. Läheltä tarkasteltaviin tekstuureihin voi käyttää vähemmän kuvan laatua huonontavia pakkauksia. (Mohi 2016.)

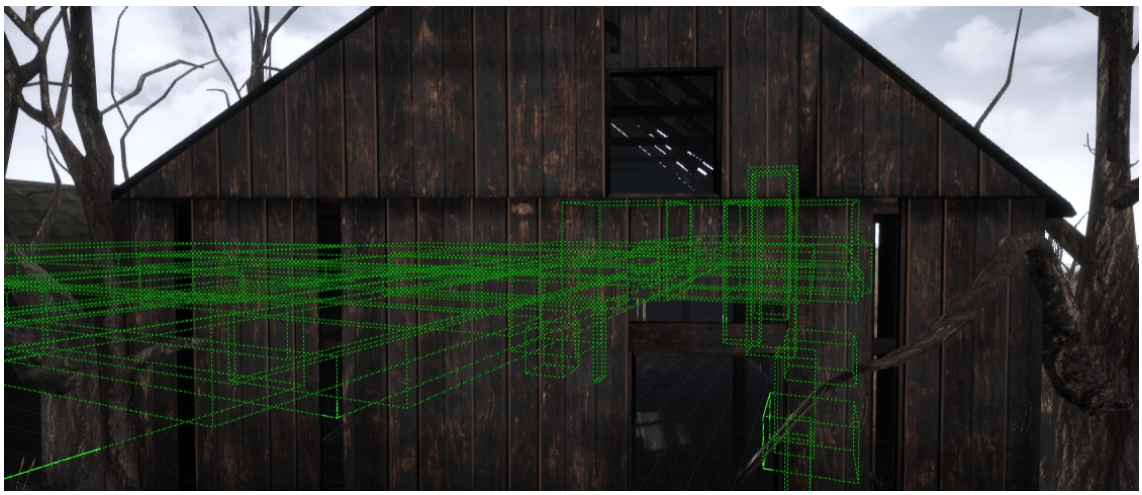
Läpinäkyvät materiaalit aiheuttavat ympäristöön myös ylipiirtämistä (engl. *overdraw*) eli saman pikselin laskemista näytölle useampaan kertaan, mikä on merkittävä kuormitustekijä GPU:lle. Läpinäkyviä materiaaleja tulisi käyttää säästeliäästi. Muissa tapauksissa ylipiirtämistä vältetään jättämällä mallien ja kameran taakse jäävät alueet renderöimättä. (O'Conor 2017.)

Kaikkia tekstuurikarttoja tehtäessä olisi myös hyvä pitää mielessä, että useimmille pelimoottoreille kahden potenssin resoluutiot (esimerkiksi 256, 512, 1024, 2048, 4096) ovat edelleen kaikkein tehokkaimpia hyödyntää. Esimerkiksi monet edellä mainituista pakkausmenetelmistä eivät tue muita kuin juuri näitä resoluutioita. (Unity Manual n.d.)

Tekstuurien lisäksi 3D-mallin materiaaleilla voi olla huomattavia vaikutuksia suorituskykyyn. Mitä useampaa materiaalia 3D-malli käyttää, sitä enemmän piirtokutsuja se CPU:lle aiheuttaa. Mitä enemmän ominaisuuksia, eli ohjeita renderöimiseen, materiaalilla on, sitä hitaammin se yleensä toimii (O'Conor 2017). Materiaalien käyttämistä shadereista kannattaa mahdollisuuksien mukaan suosia unlit-shadereita niin mobiili- kuin PC-peleissä. Useimmiten lit- ja muut monipuolisemmat shaderit ovat kuitenkin tarpeen realistisemman lopputuloksen aikaansaamiseksi. (Unreal Engine 4 Documentation n.d.)

3.4 Näkyvyysoptimointi

Näkyvyysoptimoinnissa voidaan ympäristöstä poistaa tai muokata asioita sen mukaan, miten ne on sijoitettu kameraan nähden. Usein tämä tarkoittaa mallien renderöimättä jättämistä eli seulontaa (engl. *culling*) tai LOD (engl. *level of detail*) -mallien ja mipmappien käyttämistä, jotka kaikki nopeuttavat sekä CPU:n että GPU:n suoritusta.



Kuvio 3. Vihreät kuutiot visualisoivat seulonnassa renderöimättä jätettävien mallien sijaintia.

Yksinkertaisin keino saada aikaiseksi parannuksia suorituskyvyssä on vähentää renderöitävien mallien määrää. Seulonnassa pelimoottorit jättävät kaikki kameran

näköalueen ulkopuolelle jäävät mallit renderöimättä. (Gonzalez 2017.) Pelimoottorien tarjoamista vaihtoehdoista riippuen seulontaa voidaan toteuttaa hieman eri tavoin.

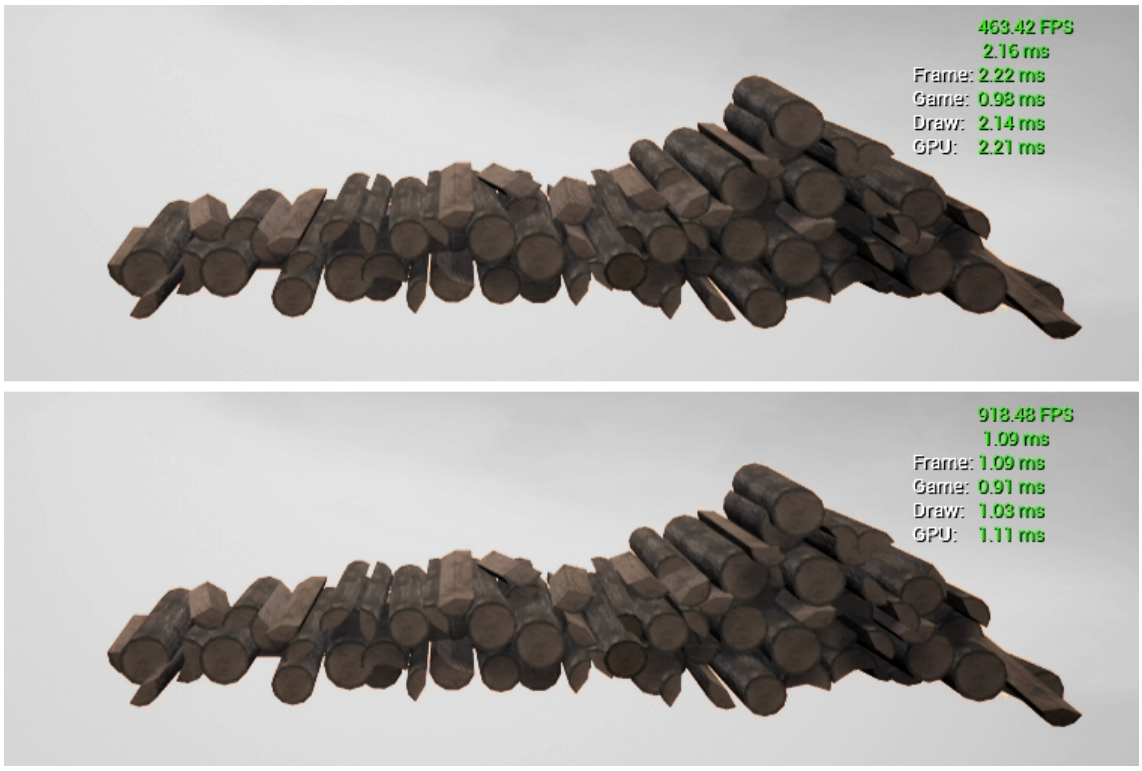
Mitä kauempana mallit ovat kamerasta, sitä pienempiä ne ovat näytöllä ja sitä vähemmän niiden eri yksityiskohdat erottuvat. Tällaisissa tilanteissa voidaan suorituskyvyn parantamiseksi käyttää LOD-malleja. (M. & Rous 2017.)

LOD-malleja tehtäessä luodaan samasta mallista yleensä pari eri versiota eri polygonimäärillä. Yleensä LOD:t luo mallintaja itse, mutta pelimoottorit saattavat tarjota niiden automaattisia generointiohjelmia. (M. & Rous 2017.) Malleista yksityiskohtaisin ja täten raskain versio on se, mitä moottori käyttää, kun mallia tarkastellaan lähietäisyydeltä. Siitä yksinkertaisempia ja kevyempiä versioita vaihdetaan edellisen tilalle aina portaittain sen mukaan, kuinka kauaksi mallista siirrytään (Cober 2017). Usein mallien polygonimäärät pienenevät jokaisella portaalla noin puoleen edellisestä (Unreal Engine 4 Documentation n.d.). Huolellisesti toteutettuna pelaaja ei huomaa eroa kaukana ja lähellä olevan mallin välillä.

Mipmappaus jatkaa LOD-mallien periaatetta tekstuurikartoilla. Suurien tekstuurikarttojen käyttö kaukana olevissa malleissa vie suoritustehoa tietokoneelta. Käyttämällä kaukana olevissa malleissa huonomman resoluution versioita samoista tekstuurikartoista säästetään suorituskykyä huonontamatta lopullisen kuvan laatua. (Gonzalez 2017.) Mipmappaus tapahtuu usein automaattisesti pelimoottorin puolesta, mutta graafikko voi tehdä karttojen eri versiot myös itse.

3.5 Niputtaminen

CPU käsittelee kerrallaan mieluummin yhden suuren mallin kuin monta pientä, koska tällöin sille syntyy käsiteltäväksi mahdollisesti vain yksi piirtokutsu. Peliympäristöjen optimoinnissa niputtaminen (engl. *batching*) tarkoittaa usean mallin yhdistämistä yhdeksi malliksi. (Unity Manual n.d.) 3D-mallien niputtaminen voidaan tehdä mallinnusohjelmassa, mutta pelimoottorit tarjoavat työkaluja niputtamiseen myös moottorin sisällä. Yksinkertaisesta konseptista huolimatta on tiettyjä asioita, jotka tulisi ottaa huomioon niputtamista toteutettaessa.



Kuvio 4. Niputtamaton (yllä) ja niputettu kasa halkoja.

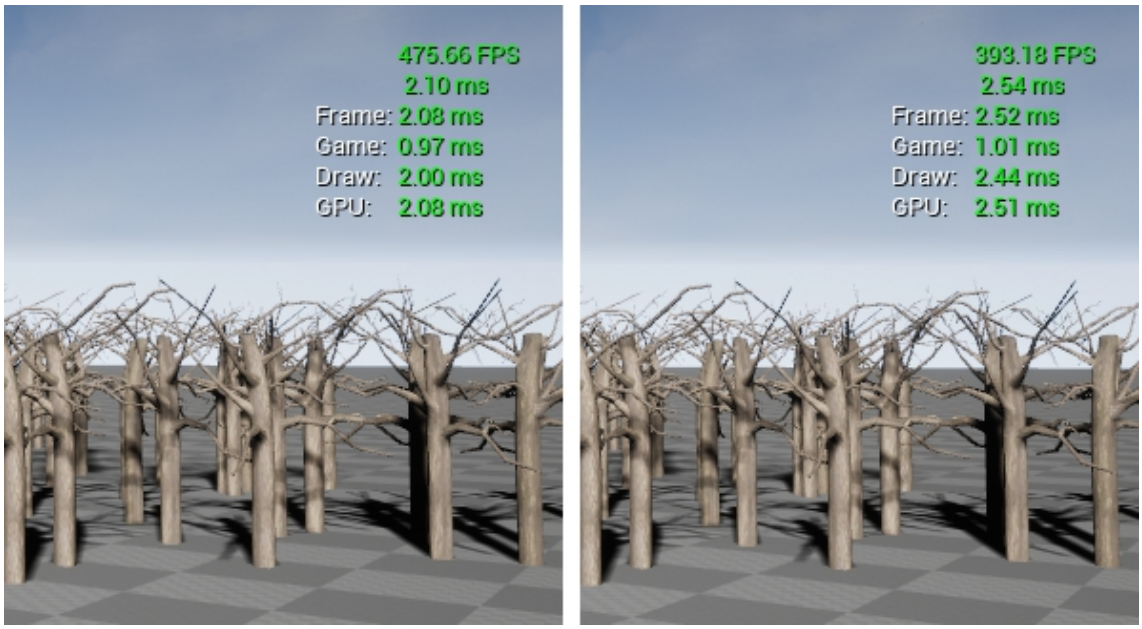
Ensinnä yhdisteltävien mallien olisi hyödyllistä sijaita ympäristössä toistensa läheisyydessä. Yhdisteltäessä niputtamiseen esimerkiksi seulomista saadaan parempia tuloksia, jos yhdistetyt mallit ovat keskitetysti lähellä toisiaan eivätkä ympäriinsä siroteltuina. Esimerkiksi jos niputtamalla yhdistetään kaksi toisistaan metrien päässä sijaitsevaa mallia, koko yhdistelmä renderöidään, vaikka vain toinen osio olisi näkyvissä. Seulonta vaatii aina mallin katoamista kokonaan kuvasta. (Cober 2017.)

Toiseksi yhdisteltäessä malleja on tärkeää edelleen tarkkailla, ettei yhdistetyn mallin verteksimäärä nouse liian korkeaksi, jolloin se hidastaa GPU:n suoritusta. Tarvittaessa tätä voi käyttää myös helppona keinona tasata laskentaa suorittimien välillä. (O'Conor 2017.)

Lopulta yhdisteltävien mallien tulisi käyttää samaa tai samoja tekstuureita ja materiaaleja. Kun sama malli käyttää kahta eri materiaalia, kokonaispiirtokutsujen määrä ei vähene. (O'Conor 2017.)

3.6 Instansointi

Ympäristöissä saatetaan usein käyttää samaa mallia monta kertaa. Esimerkiksi kokonainen metsä voi koostua vain parista erilaisesta puumallista, jotka ovat kaikki hieman eri tavalla kierrettyjä. Instansointi nopeuttaa tällaisten suurten samasta mallista koostuvien joukkojen käsittelyä (M. & Rous 2017).



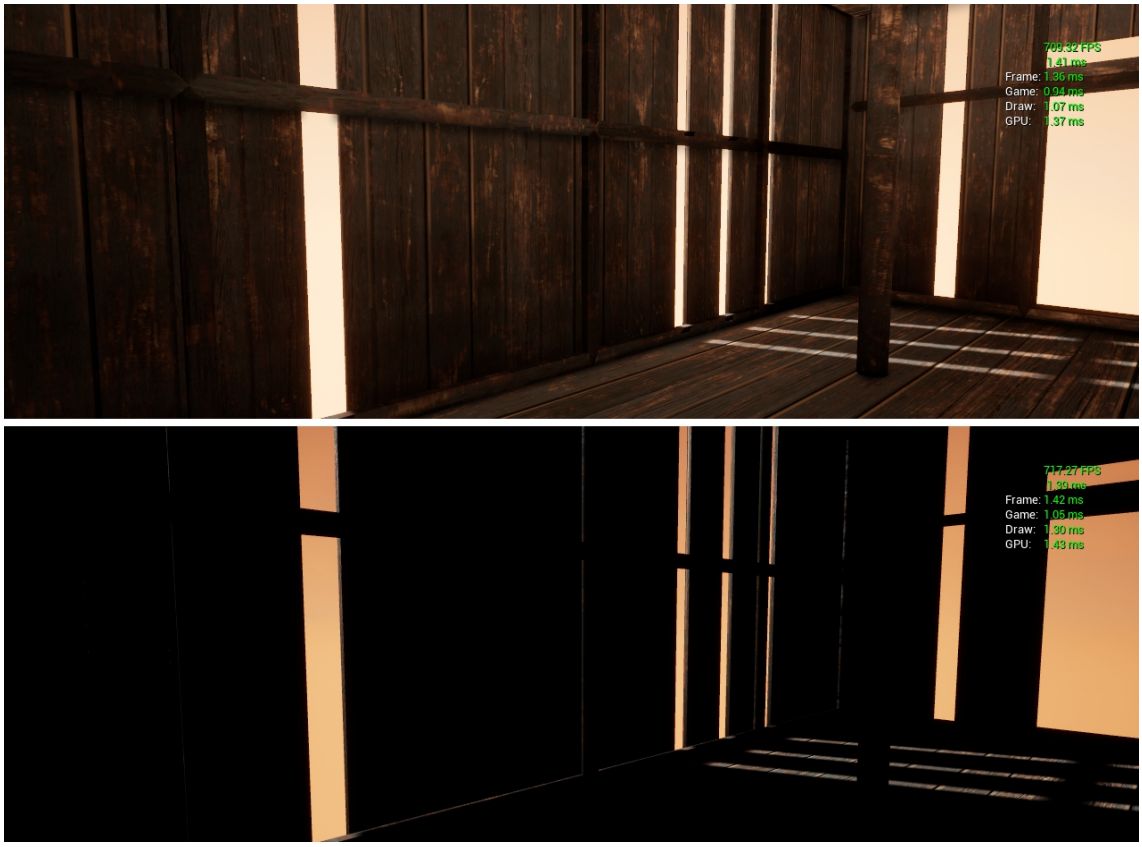
Kuvio 5. Suorituskykyero instansoidulle ja instansoimattomalle joukolle puu-malleja.

Instansointi tapahtuu yleensä pelimoottorin sisällä, sen tarjoamilla työkaluilla. Ympäristöstä merkitään kaikki samat mallit kuuluvaksi samaan ryhmään, instanssiin. Kun tulee aika renderöidä malleja, CPU ei käytä aikaa etsiäkseen jokaiselle erikseen sen materiaaleja, fysiikoita tai törmäyspintoja, vaan käsittelee kaikki mallit kerralla merkitsemällä jokaisesta yksittäisesti vain sen sijainnin, kierron ja skaalan erilliseen taulukkoon, jonka GPU voi lukea. (Świerad 2016.) Sen sijaan, että metsästä tulisi satoja tai tuhansia hyvin samanlaisia piirtokutsuja, voi CPU tuottaa saman tuloksen parhaassa tapauksessa vain yhdellä.

Koska instansoinnissa malleja ei niputtamisen tavoin yhdistetä toisiinsa, sillä ei ole vaikutuksia seulonnassa. Instansointi voidaan usein myös toteuttaa hierarkkisesti sen mukaan, kuinka kaukana mallit ovat kamerasta, jolloin se ei myöskään estä LOD-mallien käyttöä. Tällöin piirtokutsuja syntyy LOD-vaiheiden lukumäärän mukaan. (M. & Rous 2017.)

3.7 Valaistuskartat ja reaaliaikaiset valot

Valojen ja varjojen laskeminen reaaliaikaisesti ympäristön 3D-malleille vie tietokoneelta usein huomattavasti suoritustehoa. Mahdollisuuksien mukaan ne voi laskea ympäristöön jo etukäteen omaan tekstuurikarttaansa. Kuormituksen puolesta monimutkaisissa ympäristöissä on usein parempi lisätä muutama uusi tekstuurikartta CPU:n laskutoimituksiin kuin laskea kaikki valot jokaiselle mallille erikseen. Usein valaistuskarttojen käyttämisellä voidaan päästä jopa visuaalisesti parempiin tuloksiin kuin reaaliaikaisesti lasketuilla valoilla, koska tällöin GPU voi etukäteen laskea ympäristöön valon ja materiaalien värien kimpoilua ja mallien aiheuttamia varjokohtia. (Unity Manual n.d.)



Kuvio 6. Valaistuskartalla valaistu ja reaaliaikaisesti valaistu huone.

Vaikka valaistuskarttojen luominen tapahtuu käytännössä yleensä nappia painamalla pelimoottorissa, se vaatii usein onnistuakseen monenlaisien työkalujen ja asetusten läpikäyntiä. Valaistuskartat myös toimivat yleensä vain staattisille malleille, minkä vuoksi liikkuvat mallit täytyy usein valaista erikseen (Unreal Engine 4 Documentation n.d.).

Valaistuskarttoja varten 3D-malleille tulee yleensä luoda erilliset UV:t. Tekstuurikarttojen yleiset optimointikeinot, kuten esimerkiksi päällekkäisten UV-palojen käyttäminen, voivat helposti aiheuttaa ongelmia valaistuskartoissa, kun päällekkäin asetellut pinnat vastaanottavat erilaista valoinformaatiota. Valaistuskarttojen UV:t voidaan usein luoda joko pelimoottorissa tai mallinnusohjelmassa omalle UV-tasolleen. Valaistuskarttojen resoluutio on usein merkittävästi pienempi kuin tekstuurikarttojen, minkä vuoksi kartan UV-palat kannattaa asetella normaalia kauemmaksi toisistaan. Tällöin palojen valot ja varjot eivät pääse huonomman resoluution vuoksi valumaan toisiinsa. Reuna-artefaktien vähentämiseksi valaistuskarttojen laatua parantaa usein myös niiden kokoaminen mahdollisimman vähistä UV-paloista. (Unreal Engine 4 Documentation n.d.)

Ympäristön liikkuvat osaset, kuten esimerkiksi pelihahmot, valaistaan yleensä reaaliaikaisilla valoilla. Reaaliaikaisia valoja ja valaistuskartoilla valaistuja malleja käytetäänkin usein rinnakkain. Pelimoottorien tarjoamista työkaluista riippuen myös erilaiset välimuodot voivat olla mahdollisia, jolloin valaistuskartoista saatavaa valoinformaatiota voidaan hyödyntää myös liikkuvissa malleissa. Reaaliaikaisilta valoilta ei kuitenkaan aina voi välttyä, esimerkiksi käytettäessä liikkuvia valoja. Tällöin valot voi merkitä ympäristössä erikseen siten, etteivät ne piirtokutsujen minimoimiseksi vaikuta valaistuskartoilla valaistuihin malleihin. (Unreal Engine 4 Documentation n.d.)

Reaaliaikaisen valon tyypillä on vaikutuksia siihen, kuinka nopeaa niitä on käsitellä. Yleisesti ottaen pistevalot ovat kaikkein raskaimpia, suunnatut valot (directional) hieman kevyempiä ja spottivalot kaikkein keveimpiä. Myös piste- ja spottivalojen vaikutusalueella on merkitystä. Näiden lisäksi on tärkeää pitää valojen kokonaismäärä maltillisena, sillä se on suoraan yhteydessä CPU:n piirtokutsujen määrään. Usein on parempi käyttää yhtä raskaampaa valoa sen sijaan, että rakentaisi saman efektin useammalla kevyemmällä valolla. (Unreal Engine 4 Documentation n.d.)

Myös valon muut ominaisuudet vaikuttavat suorituskykyyn. Esimerkiksi staattiset, paikallaan pysyvät muuttumattomat valot ovat usein nopeimpia käsitellä, paikallaan pysyvät valot, joiden intensiteetti tai väri vaihtelevat, ovat hieman raskaampia ja täysin dynaamiset vapaasti liikuteltavat valot kaikkein raskaimpia. (Unreal Engine 4 Documentation n.d.)

Tämä pätee myös reaaliaikaisten valojen aiheuttamien varjojen suorituskyvyssä. Paikallaan pysyvistä muuttumattomista valoista syntyvät varjot ovat kevyimpiä, paikallaan pysyvistä muuttuvista valoista syntyvät varjot hieman raskaampia ja täysin dynaamisista valoista syntyvät varjot kaikkein raskaimpia renderöidä. Varjojen laatua voi myös yleensä haluttaessa säädellä tai poistaa käytöstä kokonaan. Usein reaaliaikaisten varjojen käyttö kannattaa varata vain muutamille merkittävimmille valoille suorituskyvyn säästämiseksi. (Unreal Engine 4 Documentation n.d.)

Dynaamiseen valaistukseen liittyy läheisesti myös oikean renderöintimenetelmän valinta. Usein pelimoottoreissa renderöintimenetelmäksi on valittavana joko forward- tai deferred-renderöinti.

Forward-renderöinti on lähtökohtaisesti vähemmän raskas, mutta hidastuu nopeasti, kun erilaisia objekteja, kuten 3D-malleja tai valoja, lisätään. Forward-renderöinti sopii siksi hyvin yksinkertaisiin ympäristöihin ja erityisesti yksinkertaiseen valaistukseen, jossa valaistus rakennetaan vain yhdestä tai kahdesta dynaamisesta valosta. Forward-renderöinti myös käsittelee antialiasointia ja läpinäkyvyyttä paremmin. (Shankar 2015.)

Deferred-renderöinti on lähtökohtaisesti paljon raskaampi, mutta ei hidastu juurikaan objektien lisäämisestä. Deferred-renderöinti sopii siksi puolestaan monimutkaisiin ympäristöihin ja valaistuksiin, mutta ei käsittele niin hyvin läpinäkyvyyttä ja antialiasointia kuin forward-renderöinti. (Shankar 2015.)

4 Peliympäristön toteuttamisen työjärjestys

Hyvä työjärjestys nopeuttaa projektin etenemistä, helpottaa työnkulun visualisointia, auttaa varautumaan ajoissa myöhemmin vastaan tuleviin ongelmakohtiin ja ennen kaikkea helpottaa projektin valmiiksi saattamista. Tässä luvussa esitellään yksi peliympäristöjen kehittämiselle sopiva työjärjestys ja seurataan esimerkkiympäristön luomista ja optimointia työjärjestyksen mukaisessa järjestyksessä. Työjärjestys koostuu konseptoinnista, blokkauksesta, mallintamisesta, teksturoinnista, valaistuksesta ja kameran jälkiefektien käyttämisestä. Esiteltävä työjärjestys ei ole ainoa mahdollinen, vaan se saattaa vaihdella pelistudioissa paljon. Esitetty järjestys antaa kuitenkin hyvän yleiskuvan tyypillisistä työvaiheista ympäristöjä luotaessa.

4.1 Konseptointi

Ympäristön rakentaminen lähtee yleensä liikkeelle konseptoinnista. Tällä tarkoitetaan tavallisesti nopeasti piirrettyä kuvaa, joka esittää ympäristöä, joka halutaan luoda. Konseptoinnin keinoin voidaan nopeasti luoda erilaisia vaihtoehtoja ja kehitellä ideoita siitä, millainen ympäristö voisi olla. (Raymond 2014.) Konseptoinnin pohjalta saadaan parempi käsitys siitä, minkälaisia osia ympäristö tarvitsee. Konseptin voi joko tehdä itse tai pyytää lupaa jo valmiina olevan konseptin käyttämiseen. Konsepteja voi myös olla useampia.

Muiden konseptien käyttäminen Suomen tekijänoikeuslain mukaan riippuu siitä, voidaanko lopputuloksessa edelleen katsoa olevan kyse samasta alkuperäisestä teoksesta. Niin kauan kun kyse on alkuperäisteoksen muunnelmasta, tarvitaan alkuperäisteoksen lupa oikeudenhaltijalta. Se, missä vaiheessa teoskynnys ylittyy, voi olla tulkinnanvaraista. Ohjenuorana on, että jos teos on tekijänsä henkisen luomistyön itsenäinen ja omaperäinen tulos, alkuperäisen teoksen tekijän määräysvalta ei enää päde. Tekijänoikeudella ei siis suojata ideaa tai aihetta vaan nimenomaan ilmenemismuotoa. (Tekijänoikeuslaki 1961, § 4.) Yleisesti ottaen luvan kysymistä pidetään joka tapauksessa hyvänä käytäntönä.

Hyvä konsepti tarjoaa vastauksia monenlaisiin kysymyksiin. Missä ympäristö sijaitsee? Minkälaisen kulttuurin osa ympäristö on? Minkälainen teema ja tarina ympäristöllä on?

Minkälaisista osista ympäristö koostuu? Minkälainen kiintopiste ympäristössä on? Ja mahdollisesti myös minkälaista visuaalista tyyliä ympäristö edustaa. (Galuzin 2016a.)

Konsepteja kerätessä varsinaista ympäristöä ei vielä voida optimoida, muutoin kuin valitsemalla millaisia elementtejä konseptiin otetaan. Optimoinnin hyviä puolia on kuitenkin se, että se ennemminkin mahdollistaa, kuin rajoittaa monenlaisien runsaidenkin ympäristöjen tekemisen.

Kuvio 7. Esimerkkiprojektille valittu Markus Luoteron konsepti *Swamp Lodge* (Luotero, 2015).

Kun konsepti on valittu, aihepiiriin tutustumista jatketaan etsimällä siihen liittyviä referenssikuvia ja tietoa. Kaikki informaatio mitä aihepiiristä voi kerätä ennen työn aloittamista nopeuttaa varsinaista mallinnustyötä ja lisää ympäristön todellisuuden tuntua. Referenssikuvia voidaan etsiä esimerkiksi ympäristön kuvaamasta alueesta, yksittäisistä osasista, valaistuksesta ja visuaalisesta tyylistä. (Galuzin 2016a.) Konseptia itse tehtäessä referenssikuvia etsitään usein jo ennen sitä.

Kuvio 8. Otteita tutkimuksesta.

Kuvan 4 konseptin pohjalta tehdyssä tutkimustyössä löytyi esimerkkejä siitä, miten suunnilleen kuvan kokoinen ja ikäinen lato olisi mahdollisesti laudoitettu, sekä tietoa siitä, millaista kasvustoa soilla on ja millaisissa muodostelmissa se kasvaa. Konseptien pienet yksityiskohdat voidaan helposti tulkita monella eri tavalla ja referenssien keräämisellä pyritään voi vähentää kaikkia kuvassa epäselväksi jääviä osa-alueita.

4.2 Blokkaus

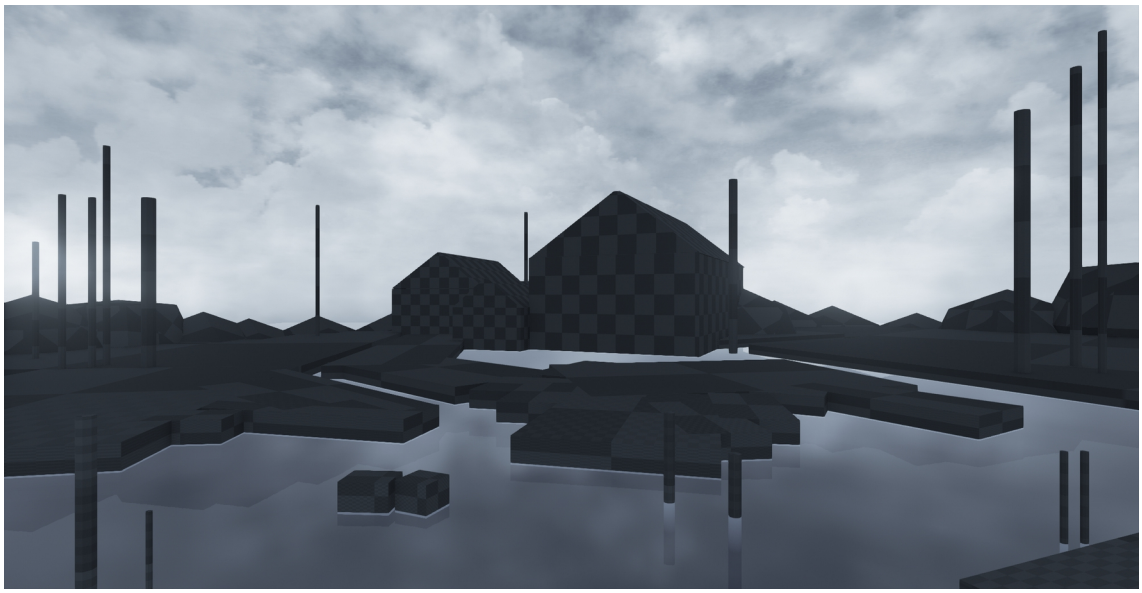
Ympäristön blokkaus tarkoittaa karkean version tekemistä ympäristöstä käyttämällä yksinkertaisia, primitiivisiä 3D-malleja, kuten kuutioita (Taylor 2014). Blokkaus tehdään yleensä suoraan pelimoottorissa käyttämällä siellä valmiiksi saatavilla olevia malleja. Tästä vaiheesta käytetään yleisesti myös englanninkielisiä termejä *grey-* tai *whiteboxing* viitaten mallissa olevan oletusmateriaalin väriin, joka esimerkiksi Unreal Enginessä on harmaa. Blokkaus on hyvin joustava prosessi, jonka voi tehdä monella tapaa, mutta tärkeintä siinä on vaiheen joustavuus ja ympäristön nopeat iteraatiot (Taylor 2014). Blokkauksen aikana ympäristöön tehdään usein mahdollisesti suuria muutoksia, joten nopealla työskentelyllä pidetään huoli siitä, että mahdollisimman vähän työtä valutetaan hukkaan.

Peliympäristöjen blokkauksessa on tärkeää määritellä ympäristön skaalat. Tarkoitus on saada aikaiseksi karkea versio ympäristöstä nopeasti, jotta sitä voidaan mahdollisimman aikaisessa vaiheessa testata ja tarvittaessa viilata niin kauan kunnes ympäristö toimii ja näyttää juuri siltä miltä sen kuuluu. (World Of Level Design 2008.)

Blokkaus auttaa suunnittelemaan ympäristöä kohti seuraavia vaiheita: ympäristön tarvitsemat mallit ja niiden mittasuhteet ja etäisyydet toisistaan ja toisiinsa verrattuna helpottavat mallinnusvaiheen työtä ja vähentävät virheiden mahdollisuuksia myöhemmissä vaiheissa. Blokkaus voi myös johtaa ympäristön hylkäämiseen tai hyllyttämiseen. Jos työ alkaa blokkausvaiheessa vaikuttamaan liian massiiviselta, yksinkertaiselta tai haastavalta, se on tässä vaiheessa vielä helppo jättää sivuun.

Blokkausvaiheen jälkeen ympäristöä aletaan rakentamaan sen päälle. Monissa pelimoottoreissa on mahdollisuus viedä blokatut muodot ulos esimerkiksi fbx-muodossa. Tällöin ne voidaan viedä suoraan haluttuun 3D-mallinnusohjelmaan, missä yksityiskohtaisempi 3D-malli voidaan luoda suoraan blokkauksen päälle, säilyttäen halutut mitat. Mallien valmistuessa niitä vastaavat palikat voidaan yksinkertaisesti korvata pelimoottorissa ja näin seurata ympäristön valmistumista askel askeleelta ilman, että kokonaiskuva kuitenkaan katoaa.

Blokkausvaiheessa voi jo aloittaa varautumisen tuleviin optimointikeinoihin. Isommilla blokeilla voidaan esimerkiksi jakaa ympäristöä siten, että pienempiä yksityiskohtia jää seulonnassa ajoittain niiden taakse, jolloin näytölle ei tulla renderöimään kerralla liikaa. Blokkausvaiheessa usein ymmärrys siitä missä paikoissa voidaan käyttää samoja malleja (instansointi) ja missä tiloissa malleja voi mahdollisesti yhdistellä (niputtaminen) usein tarkentuu. Blokattaessa myös ensimmäistä kertaa nähdään tarkemmin mallien etäisyys kamerasta, joka taas auttaa alustavasti arvioimaan LOD-mallien ja mipmappien tarvetta paremmin. Myös valaistuksen tarvetta ja rakentamista voidaan alustavasti testata blokkauksen aikana.



Kuvio 9. Valmis blokkaus Unreal Enginessä.

Valitun esimerkkiympäristön blokkauksessa on merkitty keskiosan rakennukset, merkittävimmät puut, pusikot ja suomättäiden rajat. Blokkauksesta selvisi LOD-mallien ja instansoinnin tarve erityisesti suomättäiden kasvillisuudelle ja puille. Seulonnalle ympäristössä ei ole paljoa mahdollisuuksia, muutoin kuin lähestyttäessä keskiosan latoja, joten isompien näköesteiden lisäämistä voi harkita. Blokkauksen ohessa määriteltiin myös kameran asetukset. Oikean kuvasuhteen ja kuvakulman löytyminen auttoi myös merkittävästi sijoittelemaan rakennuksia ja selvittämään niille realistiset koot. Kameran ansiosta voi ympäristön rakentuessa jatkuvasti myös tehdä vertailua takaisin konseptiin.

Ympäristön muotojen lisäksi tässä blokkauksessa on luotu myös nopeat version konseptin pilvisestä, tasaisesta valaistuksesta ja vedestä, jotka ovat kaksi suurta elementtiä kuvassa 3D-mallien ohella. Kokeilemalla luoda ympäristöön tarvittavia tärkeitä komponentteja nopeasti välttyy ikäviltä yllätyksiltä myöhemmin projektin aikana (Taylor 2014). Jos toinen tai molemmat näistä elementeistä osoittautuisivat jo tässä vaiheessa ja tällä tasolla liian haastaviksi, projektia voisi vielä helposti vaihtaa menettämättä juurikaan arvokasta työtä ja siihen käytettyä aikaa.

Pelimoottorit tarjoavat usein työkaluja blokkauksen tekemiseen. Yleensä tarjolla on muutamia eri primitiivejä, joita voi liikutella, pyöritellä ja skaalata, joiden verteksejä, särmiä ja tahkoja voi muokata halutun muodon aikaansaamiseksi. Kovin monimutkaisia malleja tarjolla olevilla työkaluilla ei ole yleensä mahdollista tehdä.

4.3 3D-mallintaminen

Mallintaminen aloittaa viimeistään projektin tuotantovaiheen. Esituotantoa (konsepti, referenssit ja blokkaus) luonnehtivat suuret muutokset ja nopeat iteraatiot ympäristöstä. Näiden jälkeen muutosten ja vaihdosten tekeminen projektissa on usein hidasta ja kokonaisuuden kannalta riskialtista. Muutosten tekemiseltä ei aina voi välttyä myöhemmissä vaiheissa, mutta huolellisesti toteutettu esituotanto karsii pois kaikkein kookkaimmat ja haitallisimmat ongelmat ja siksi siihen tulisi käyttää sen tarvitsema aika. Esituotantovaiheen päättyessä on tärkeää tarkistaa itseltään, että projektin tavoitteet ja lopputulos ovat piirtyneet selkeänä mieleen. (Galuzin 2016) Mallintamisen alkaessa tulisikin projektissa ympäristön suunnittelun sijaan siirtyä optimoinnin hallitsemiseen ja toteuttamiseen.

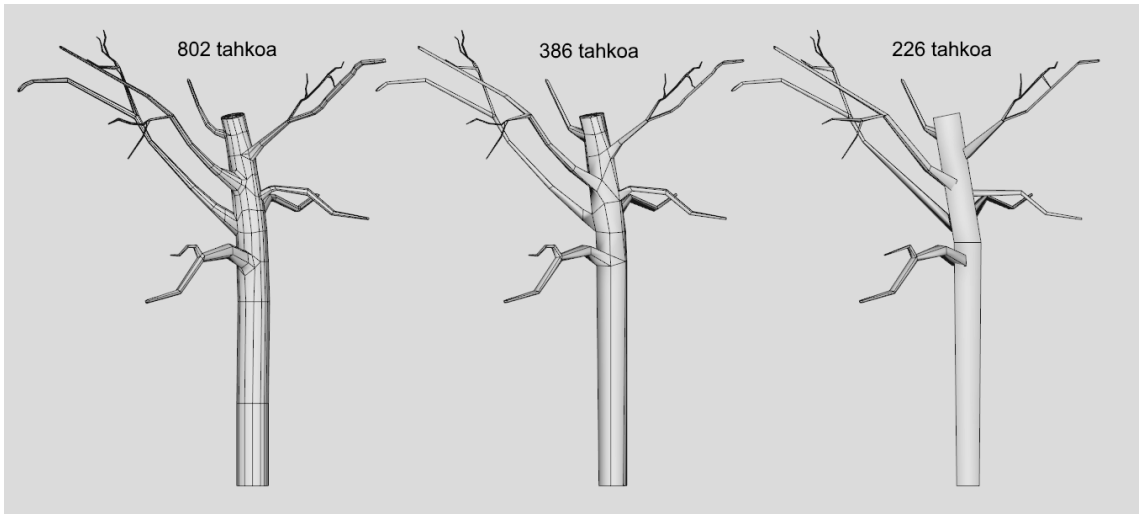
Sopivasta polygonimäärästä ei ole yhtä oikeaa vastausta saatavilla. Pelimoottoreissa on eroa siinä, kuinka hyvin ne käsittelevät polygoneja, ja peleissä ja ympäristöissä on eroja esimerkiksi siinä, kuinka monta mallia näytölle piirtyy kerralla. Läheltä tai merkittävällä paikalla olevat erikoismallit voivat oikeuttaa useampien polygonien käytön, kuin sivummalle jäävät mallit. Erikoismallien ulkopuolella kannattaa pyrkiä tasalaatuisiin malleihin, jolloin ne parhaiten tukevat ympäristön kokonaisuutta. Mallinnettaessa siluetin silmällä pitäminen voi auttaa löytämään sopivat polygonimäärät ja virheiden sattuessa puhdas topologia mahdollistaa polygonien määrän helpon vähentämisen. (Thacker 2016.) Samalla tavoin onnistunut topologia helpottaa myös esimerkiksi LOD-mallien luomista.

Mallinnettaessa myös mallinnusjärjestyksen tiedostaminen voi helpottaa vaiheen kulkua. Ulkotiloihin sijoittuvissa ympäristöissä maaston muodot määrittelevät paljon, miten eri mallit asettuvat ympäristöön ja miten katse liikkuu ympäristössä. Sisätiloissa tähän vaikuttavat esimerkiksi seinät ja tilojen asettelu. Suuria maastoja varten monet pelimoottorit tarjoavat työkaluja maaston luomiseen ja sen muotojen kohottamiseen ja laskemiseen. Maaston jälkeen suuremmat mallit vievät kuvasta seuraavaksi eniten tilaa ja vaikuttavat aina pienempien mallien asetteluun niiden ympärillä. Kasvillisuus tai muut pienet yksityiskohdat lisätään usein viimeksi niille vapaaksi jääneisiin tiloihin. Pienemmät mallit vievät usein kuvasta vähemmän tilaa mutta ovat monilukuisempia, joten niiden sijoittaminen viimeiseksi auttaa määrän kontrolloinnissa. Korvaamalla

malleja yksitellen blokkauksen tilalle säilytetään jatkuvasti ympäristön kokonaisuus ja nähdään ajoissa, jos tilalle tuodut palaset eivät loksahdakaan yhteen.

Kuvio 10. Valmis mallinnusvaihe.

Esimerkkiympäristössä mallinnus aloitettiin maaston muodoista, minkä jälkeen siirryttiin kahteen keskellä olevaan vanhaan latoon, joista puihin ja lopulta muuhun kasvillisuuteen. Järjestyksessä toimiminen helpotti esimerkiksi valitsemaan montako erilaista puumallia ympäristöön olisi tarpeellista tehdä riittävän vaihtelun aikaansaamiseksi. Puita kiertämällä eri suuntiin voidaan saada aikaiseksi erilaisia siluetteja, jolloin kaikkien puiden ei tarvitse olla uniikkeja. Pienempien kasvien muodot tukeutuvat enemmän tekstuurien läpinäkyvyyteen, jonka vuoksi niiden lopullinen paikoitus selviää vasta teksturoidessa.



Kuvio 11. Puun eri LOD-tasot.

Puita on ympäristössä paljon, ja niiden etäisyydet vaihtelevat suuresti, jonka vuoksi puista luotiin kolme LOD-versiota Maya-mallinnusohjelmassa. Pienempää ruohokasvillisuutta on myös ympäristössä runsain määrin, mutta niissä jo läheltä tarkastellessa polygonimäärä on niin pieni, että LOD-ryhmiä ei luotu. Kasvillisuuden optimointi tulee tukeutumaan enemmän tekstuurien mipmappaukseen. Ladot ovat ympäristön keskipiste ja myös muodoiltaan riittävän yksinkertaisia oikeuttaakseen samoin vain kaksi LOD-mallia.

Kuvio 12. Mallinnusvaiheen profilointi.

Mallinnusvaiheen lopulla kuvataajuus on noin 250 FPS. Sekä draw eli CPU että GPU ovat noin 3,9 ms:n lähistöllä, eli molemmat tekevät työtä tasavertaisesti. Puut ja kasvillisuus on instansoitu ja kameran näköalueen ulkopuolelle jäävät mallit on seulottu pois. LOD-mallit on otettu käyttöön sopivilla etäisyyksillä. Myös kasvillisuus on jo alustavasti niputettu pieniksi kokonaisuuksiksi mättäittäin. Keskiosan ladot koostuvat vain kahdesta osioista (katto ja seinät) niille suunniteltujen tekstuurien mukaisesti. Koska ympäristö on tarkoitettu esittely- eikä pelikäyttöön, törmäyspinnat on voitu jättää kokonaan pois.

Esimerkkiympäristön profiloimiseen on käytetty Windows 10 -tietokonetta AMD FX-8350 CPU:lla ja Nvidia GeForce GTX 1060 GPU:lla. Emolevy on ASUSTek M5A99X EVO ja RAM-muistin määrä 24 Gt. Unreal Enginestä esimerkkiympäristössä on käytetty versiota 4.17.2.

4.4 Teksturointi

Huolellinen teksturointi vaatii aikaa ja tarkkuutta. Tekstuurien luominen ja UV-karttojen avaaminen ja asettelu on usein hidasta puuhaa. Huonojen tekstuurien käytön korvaaminen hyvin optimoiduilla versioilla myöhemmin voi käytännössä vaatia koko prosessin tekemistä kokonaan uudelleen, jonka vuoksi se kannattaa tehdä alusta alkaen hyvien käytäntöjen mukaisesti.

Kuvio 13. Valmis teksturointivaihe.

Teksturointi aloitettiin mallinnuksen tavoin suuremmista osa-alueista siirtyen pienempiä kohti. Ympäristön maasto, ladot ja puut käyttävät kaikki tile-tekstuureja, joista pelimoottori laskee automaattisesti useampia mipmapattuja tasoja. Lähimmän tason tile-tekstuurit on tehty 2048*2048 pikselin tarkkuudella maastoa lukuun ottamatta, jossa resoluutio on 512*512. Maasto jää suurilta osin peittoon kasvillisuuden alle, mutta hyvä pohja vähentää tarvetta joka kolon piilottamiselle ja pieni tile-tekstuuri on maastoissa kaikkein edullisin hyödyntää. Koska näissä tekstuureissa ei ole läpinäkyvyyttä ollenkaan, niissä voitiin käyttää DXT1-pakkausta.

Tile-tekstuurit myös mahdollistavat UV:iden runsaan päällekkäisen käytön. Mahdollisimman vähien UV-saarekkeiden käyttö, pintaa vääristämättä, auttaa suorituskykyä myös. Koska UV-palat ovat päällekkäisiä, tekstuureille täytyi luoda myös erillinen kanava muun muassa seuraavan vaiheen valojen laskemista varten.

Kasvillisuus on aseteltu lopulliselle paikalleen vasta nyt tekstuurien valmistumisen myötä. Kaikki ympäristön kasvit on koottu samaan 2048*2048 pikselin atlakseen, joka

mipmapataan myös useisiin versioihin. Kasvillisuuden muoto perustuu pitkälti tekstuurikarttojen läpinäkyvyyteen, mikä on aina riski peliympäristöissä ylipiirtämisen vuoksi. Kasveissa läpinäkyvyyttä tarvitaan polygonien vähentämiseksi, mutta mallinnuksessa pyrittiin silti ottamaan huomioon läpinäkyvien alueiden minimoiminen. Jos suorituskyky tippuu GPU:n puolelta merkittävästi, voidaan ylipiirtoa vähentää helposti lisäämällä polygoneja ja LOD-ryhmiä kasvien malleihin. Kasvien UV:t on helppo asetella niiden mallien yksinkertaisen muodon ja vähäisen polygonimäärän vuoksi ja läpinäkyvyyden käyttämisen vuoksi tekstuuriatlaksen pakkaamiseen käytettiin DXT5-pakkausta.

Teksturointivaiheessa ympäristöön luotiin myös uusi heijastava vesimateriaali. Heijastus on luotu Unreal Enginen Planar Reflection -ominaisuuden avulla, joka on suorituskyvyltään erittäin kallis, mutta vaihtoehtoja realistisempi. Heijastus on aina suorituskyvylle raskasta, joten siihen täytyy varautua hyvin. Hyvä optimointi tähän asti on mahdollistanut suorituskyvyn käyttämisen tässä kohtaa, veden ollessa ympäristössä merkittävä elementti.

Kuvio 14. Teksturointivaiheen profilointi.

Teksturoinnin jälkeen kuvataajuus on noin 115 FPS, eli ympäristön suorituskyvyssä on tapahtunut merkittävä pudotus. Suurimmaksi osaksi pudotus johtuu heijastavasta vesimateriaalista, joka tiputti FPS:n 250:stä 135:een. CPU:n nopeus on 6,7 ms ja GPU:n 8,6.

Ympäristön luomisen työjärjestys ei ole aina täysin lineaarinen. Suorituskykyä seurataan koko matkan ajan, jotta on helpompaa huomata missä kohtaa se putoaa ja

korjata asia ajoissa. Lopullinen tulos paljastuu kuitenkin vasta kaiken ollessa valmis. Tässä vaiheessa esimerkiksi CPU:n ja GPU:n väliseen laskentaeroon ei kannata vielä puuttua, koska valaistuskarttojen tekeminen yleensä keventää GPU:n taakkaa. Hyvä optimointi alusta alkaen helpottaa lisäoptimointia myös myöhemmin, ja jos suorituskykyä jää loppuvaiheessa vielä käytettäväksi, voidaan aikaisempiin vaiheisiin palata myös lisäämään yksityiskohtia.

4.5 Valaistus

Valaistus on peliympäristöjen tärkein yksittäinen tunnelmanluoja. Valaistus määrittelee, miten pelaaja kokee ympäristön, mihin katse keskittyy ja miten immersiivisenä maailma välittyy. Huonolla valaistuksella voidaan vielä helposti pilata hyvä mallinnus- ja teksturointityö. (Pluralsight 2014.)

Hyvä valaistus on usein yksinkertainen. Suurin osa ympäristön valoisuudesta tulee yhdestä selkeästi määriteltävästä kohteesta, kuten auringosta tai spottivalosta, jota muut ympäristön valot tukevat. Tärkeimmän valonlähteen hiominen mahdollisimman pitkälle ennen muiden valojen lisäämistä auttaa valojen määrää pysymään maltillisena. Liian monien voimakkuudeltaan kilpailevien valojen käyttämisellä on usein katsojalle hämmentävä vaikutus. Hyvät referenssit auttavat myös valaistuksen rakentamisessa. (Betancourt 2017.)

Peliympäristön valaistus ei kuitenkaan aina synny vain asettelemalla valoja ympäristöön. Yleensä valoja tukemassa tai niihin vaikuttamassa voi olla esimerkiksi sumua, taivasvaloa ja emissiivisiä materiaaleja, jotka kaikki vaikuttavat ympäristön valoisuuden määrään ja tunnelmaan. Näiden komponenttien työstäminen yhdessä samanaikaisesti helpottaa kokonaisuuden hallitsemisesta.

Kuvio 15. Valmis valaistusvaihe.

Valaistusta lähdettiin rakentamaan suoraan blokkausvaiheessa luodun alustavan valaistuksen päälle. Blokkauksesta lähtien ympäristön valaistus on ollut lähtöisin taivasvalon luomasta tasaisesta valosta. Taivasvalon hienosäätäminen ensin tarjosi hyvän pohjan valaistuksen rakentamiselle.

Esimerkkiympäristön valmis valaistus koostuu neljästä eri komponentista. Tärkeimmän taivasvalon lisäksi ympäristöön lisättiin himmeä suunnattu valo, joka auttaa tuomaan latojen katto-osia paremmin esille, proseduraalinen pilvinen taivasmateriaali, joka ei juurikaan luo valoa mutta tuo hieman hehkua mallien siluettien ympärille, ja volumetrinen sumu, joka luo kuvaan etäisyyden tunnetta. Ympäristön valonlähteiden vähäisyyden vuoksi valaistus on helppo pitää koossa.

Reaaliaikaisia valoja ympäristössä ei ole ollenkaan, koska kaikki valot on laskettu valaistuskarttoihin. Koska ympäristössä käytettiin runsaasti tile-tekstuureja, suurimmalle osalle malleista luotiin pelimoottorissa automaattisesti erillinen UV-kanava valaistuskartan käytettäväksi. Kullekin mallille saa erikseen määriteltyä resoluution valaistuskartasta, mikä helpottaa ympäristön eri kohteiden valaistuksen priorisointia.

Taulukko 1. Valaistusvaiheen profiloinnin tulokset.

Valon tyyppi ja renderöintimenetelmä	**FPS**	**CPU**	**GPU**
Lasketut valot + forward-renderöinti	125	7.3	7.8
Lasketut valot + deferred-renderöinti	120	7	8.2
Reaaliaikaiset valot + forward-renderöinti	100	9.5	9.8
Reaaliaikaiset valot + deferred-renderöinti	95	9.9	10.5

Esimerkkiympäristön valaistuksen rakentamisen loppupuolella testattiin vielä millainen yhdistelmä toisi parhaan suorituskyvyn. Vertailussa laskettiin FPS:n, CPU:n ja GPU:n arvot valaistuskarttojen ja reaaliaikaisen valaistuksen sekä forward- ja deferred-renderöinnin eri yhdistelmien aikana. Tähän asti opinnäytetyössä on ollut käytössä reaaliaikaiset valot ja deferred-renderöinti.

Odotetusti reaaliaikaiset valot olivat valaistuskarttoihin laskettuja valoja raskaampia. Paras tulos saatii käyttämällä valaistuskarttoja ja forward-renderöintiä. Toisin kuin reaaliaikaisessa valaistuksessa, valaistuskarttojen käytössä eri renderöintimenetelmillä ei pitäisi olla suuria eroja. Forward-renderöintimenetelmä antoi kuitenkin hieman paremman suorituskykytuloksen, luultavasti johtuen sen kyvystä käsitellä tehokkaammin ympäristön kasvillisuudesta tulevaa ylipiirtoa. Deferred-renderöintimenetelmää käytettäessä CPU:n ja GPU:n laskenta on epätasapainossa, joten näitä tasaamalla olisi voitu saada hieman parempiakin tuloksia.

Teksturointivaiheesta FPS on parantunut 115 FPS:stä 125 FPS:ään. Myös CPU ja GPU ovat paremmin tasoissa valaistuskarttojen vähennettyä GPU:lle tulevaa laskentaa. Nousu olisi ollut luultavasti korkeampi, ellei valaistusvaiheen aikana olisi lisätty myös volumetrista sumua.

4.6 Jälkiefektit

Jälkiefekteillä viimeistellään ympäristö. Ne mahdollistavat reaaliaikaisen kuvan muokkaamisen hieman samaan tapaan, kuin valokuvan muokkaamisen kuvankäsittelyohjelmassa. Jälkiefekteillä voi säädellä monipuolisesti useita erilaisia kuvan ominaisuuksia, kuten esimerkiksi valon hehkumista, AO:ta, värien säätelyä, antialiasointia ja syväterävyyttä (Unreal Engine 4 Documentation n.d.)

Jälkiefektit vievät yleensä merkittävästi suorituskykyä, mutta myös eroavaisuuksia efektien raskauksissa löytyy. Yleisesti ottaen esimerkiksi AO:n lisääminen on raskas efekti verrattuna värien ja valon hehkun lisäämiseen. Kaikki riippuu kuitenkin käyttöön otetun efektin voimakkuudesta ja metodeista efektin taustalla, sillä lähes kaikille efekteille on yleensä saatavilla useita suorituskyvyltään ja laadultaan erilaisia vaihtoehtoja.

Kuvio 16. Valmis jälkiefektivaihe.

Merkittävin lisäys tässä vaiheessa oli forward-renderöintimenetelmän tarjoaman MSAA-antialiasoinnin käyttöönotto, joka näkyy erityisesti kasvillisuuden renderöinnissä. MSAA on raskas mutta erittäin laadukas antialiasointimenetelmä. Antialiasoinnin lisäksi ympäristöön lisättiin vain muutamia pienempiä efektejä, kuten valon hehkua, vinjetointia ja värien säätelyä.

Kuvio 17. Jälkiefektivaiheen profilointi.

Jälkiefektien lisäämisen jälkeen kuvataajuus on noin 100 FPS ja CPU:n laskenta on noin 7,5 ms ja GPU:n 9,5 ms. Koska FPS on vielä näin korkea, voidaan työjärjestyksessä palata aikaisempiin vaiheisiin ja tehdä ympäristöön parannuksia niin kauan kunnes toivottu FPS saavutetaan. Parannuksien tekemisen ohella voidaan myös CPU:n ja GPU:n laskennan tasoa tasata.

Kuvio 18. Valmis ympäristö.

Jälkiefektivaiheesta saadun suorituskykytuloksen jälkeen ympäristöön lisättiin vielä kasvillisuuden määrää, polygoneja puu- ja latomalleihin, parannettiin tekstuurien laatua ja monipuolistettiin kasvillisuuden materiaalia lisäämällä esimerkiksi tuulessa huojumisen animaatiota. Optimoinnin ansiosta kaikki halutut komponentit saatiin mukaan ja FPS lopulliselle ympäristölle oli keskimäärin 90. Tästäkin suorituskyvystä on vielä paljon liikkumavaraa PC-peleille tavoitteelliseen 60 FPS:n. On kuitenkin otettava huomioon, että peleissä muutkin ominaisuudet ja toiminnallisuus kuin grafiikka vievät suorituskykyä, jonka vuoksi hieman liikkumavaraa voi olla hyvä jättää.

5 Pohdinta

Peliympäristöt ovat ikkunoita toisiin maailmoihin, ne syventävät pelissä kerrottua tarinaa ja parhaimmillaan kertovat ja sisältävät itsessään useita pieniä tarinoita lisää. Peliympäristöt sisältävät monia eri komponentteja, ja kilpailu kauneimmista tai realistisimmista ympäristöistä alalla on kovaa. Hyvä optimointi mahdollistaa aina runsaimman mahdollisen lopputuloksen mille tahansa kohdelaitteelle, ja hyvä työjärjestys edesauttaa projektin valmistumista ja vähentää ongelmien syntymistä projektin aikana. Tässä projektissa tavoitteenani oli oppia mahdollisimman paljon optimoinnista ja testata työjärjestyksessä toimimisen hyötyjä. Molemmat tavoitteet täyttyivät.

Esitelty työjärjestys antoi hyvän pohjan opinnäytetyön ohessa luomalleni peliympäristölle. Työskentelyni sujui mukavasti, eikä suuria yllätyksiä matkan varrella tullut. Optimointimenetelmistä muutamat olivat entuudestaan tuttuja, mutta suurin osa aivan uusia, mikä helpottaa ja luo uusia näkökulmia myös tulevien ympäristöjen suunnitteluun ja luomiseen. Peliympäristöjen optimointi kehittyy jatkuvasti, ja työstä saadulla pohjalla sen seuraamista ja tutkimista on mielekästä jatkaa.

Tiesin, että työjärjestyksen ja optimointitekniikoiden lisäksi halusin oppia käyttämään lisää Unreal Engine -pelimoottoria, jonka valitsin yhdeksi työkalukseni tähän opinnäytetyöhön. Uuden ohjelmiston käyttöönotto opinnäytetyön tekemisen ohella aiheutti ajoittaisia ongelmia tiettyjen ominaisuuksien löytämisessä ja käyttämisessä, mutta niiden kautta myös taidot ohjelmiston kanssa kasvoivat. Unreal Engine on monipuolinen ohjelmisto, joten opittavaa on edelleen paljon jäljellä.

Pelimoottorit kehittyvät nopealla tahdilla, ja jatkuvasti käytettäväksi tulee uusia ominaisuuksia ja samalla niihin liittyviä uusia optimointitapoja. Tietoa etsiessä huomasin nopeasti, kuinka nopeasti lähteet vanhentuvat ja ovat ristiriidassa uudemman tiedon kanssa. Myös monet asiat, jotka ennen täytyi huolehtia itse, kuten LODit, instansointi, seulominen, niputtaminen ja mipmapit, tehdään nyt pelimoottoreissa usein automaattisesti. Edelleen on kuitenkin hyödyllistä olla mahdollisimman laaja-alaisesti tietoinen automaattisestikin toteutetuista

optimointimenetelmistä, sillä ne usein vaativat toimiakseen jonkinlaista aktiivistakin valmistelua.

Olisin voinut tehdä vielä enemmän suorituskyvyn vertailua optimoitujen ja optimoimattomien ympäristöversioiden välillä esimerkiksi instansointiin, seulontaan ja niputtamiseen liittyen. Olisi myös ollut kiinnostavaa tutkia enemmän, miten ympäristön rakenteen suunnittelulla (level design) voidaan vaikuttaa optimointiin muutoin kuin esimerkiksi asettelemalla näköesteitä seulonnan edesauttamiseksi.

Opinnäytetyön tekemisen aikana huomasin myös, että projekteja tehdessä jonkin tietyn näkökulman tai opittavan asian kohteeksi valitseminen auttaa pitämään projektin mielenkiintoisena ja merkityksellisenä loppuun asti. Saatu hyöty on tällöin enemmän kuin vain kuva portfoliossa ja usein myös lopputulos odotettua parempi.

Lähteet

Ahearn, Luke 2008. 3D Game Environments: Create Professional 3D Game Worlds. Kustannuspaikka: Elsevier.

Betancourt, Gabe 2014. Learning Lighting For Video Games. <https://80.lv/articles/learning-lighting-for-video-games/ > (luettu 30.10.2017).

Cober, Bob 2017. UE4 - Overview of Static Mesh Optimization Options. <http://www.casualdistractiongames.com/single-post/2017/01/07/UE4---Overview-of-Static-Mesh-Optimization-Options> (luettu 24.9.2017).

Galuzin, Alex 2016a. How To Plan Level Designs And Game Environments In 11 Steps <http://worldofleveldesign.com/categories/level_design_tutorials/how-to-plan-level-designs-game-environments-workflow.php> (luettu 23.10.2017).

Galuzin, Alex 2016b. Making Difficult Level Design Decisions <http://www.worldofleveldesign.com/categories/productivity_goals/making-difficult-level-design-decisions.php> (luettu 23.10.2017).

Gesota, Rudra 2016a. A Simple How To Guide For Graphics Optimization In Unity. <http://www.theappguruz.com/blog/graphics-optimization-in-unity> (luettu 10.10.2017).

Gesota, Rudra 2016b. How to Make Your Games Run Superfast by Using Draw Call Reduction. <http://www.theappguruz.com/blog/learn-draw-call-reduction-and-make-your-games-run-superfast> (luettu 10.10.2017).

Gonzalez, Jonathan 2017. Maximizing Your Unity Game's Performance. <https://cgcookie.com/articles/maximizing-your-unity-games-performance> (luettu 9.10.2017).

Ivanov, Ivan-Assen 2006. Practical Texture Atlases. <https://www.gamasutra.com/view/feature/130940/practical_texture_atlases.php> (luettu 20.10.2017).

Klappenbach, Michael 2017. Understanding and Optimizing Video Game Frame Rates. <https://www.lifewire.com/optimizing-video-game-frame-rates-811784> (luettu 23.9.2017).

Luotero, Markus 2015. Kuva. Swamp Lodge. <https://www.artstation.com/artwork/KE5Z4> (katsottu 6.11.2017).

McKinley, Michael 2005.The Game Artist's Guide to Maya. Kustannuspaikka: Maya Press.

M., Errin & Rous, Jeff 2017. Unreal Engine 4 Optimization Tutorial, Part 2. <https://software.intel.com/en-us/articles/unreal-engine-4-optimization-tutorial-part-2> (luettu 23.9.2017).

Mohi, Ahmeg 2016. Why You Should Use "Advanced Texture Type" More Part 2 –

Unity3D. <http://2024studios.blogspot.fi/2016/01/why-you-should-use-advanced-texture.html> (luettu 10.10.2017).

O'Conor, Keith 2017. GPU Performance for Game Artists. <http://fragmentbuffer.com/gpu-performance-for-game-artists/> (luettu 23.9.2017).

Pluralsight, 2014. Light Up Your World: How Lighting Makes All The Difference For Games. <https://www.pluralsight.com/blog/film-games/understanding-the-importance-of-lighting-for-games> (luettu 30.10.2017).

Polycount Wiki n.d. Polygon Count. <http://wiki.polycount.com/wiki/Polygon_Count> (luettu 10.10.2017).

Raymond, Jon 2014. Concept Art: What Is Concept Art And Why Is It Important?. <http://artistryingames.com/concept-art-concept-art-important/> (luettu 19.10.2017).

Shankar, Darshan 2015. A Brief Overview of Lighting In Unity 5. <https://dshankar.svbtle.com/lighting-in-unity-5> (luettu 2.11.2017)

Świerad, Oskar 2016. UE4 Optimization: Instancing. <https://www.youtube.com/watch?v=oMIbV2rQO4k > (katsottu 24.9.2017).

Taylor, James 2014. Creating A Simple Level Blockout. <https://www.methodj.com/creating-a-simple-level-blockout/> (luettu 16.9.2017).

Tekijänoikeuslaki 1961. § 4. <http://www.finlex.fi/fi/laki/ajantasa/1961/19610404> (luettu 24.9.2017)

Thacker, Jim 2016. Five Expert Tips To Improve Your Games Environment Art. <https://www.gnomon.edu/blog/five-expert-tips-to-improve-your-games-environment-art> (luettu 23.10.2017).

Unity Manual n.d. Art Asset Best Practice Guide. <https://docs.unity3d.com/Manual/HOWTO-ArtAssetBestPracticeGuide.html> (luettu 10.10.2017).

Unity Manual n.d. Optimizing Graphics Performance. <https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html> (luettu 23.9.2017).

Unreal Engine 4 Documentation n.d. Shading Models. <https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/MaterialProperties/LightingModels/> (luettu 10.10.2017).

Unreal Engine 4 Documentation n.d. Ligthmass Global Illumination. <https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/Lightmass/index.html> (luettu 24.9.2017).

Unreal Engine 4 Documentation n.d. Performance Guidelines for Artists and Designers. <https://docs.unrealengine.com/latest/INT/Engine/Performance/Guidelines/> (luettu 23.9.2017).

Unreal Engine 4 Documentation n.d. Unwrapping Uvs for Lightmaps. <https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/Lightm

apUnwrapping/index.html> (luettu 9.10.2017).

World Of Level Design 2008. Art Of Blocking In Your Map. <http://www.worldofleveldesign.com/categories/level_design_tutorials/art_of_blocking_in_your_map.php> (luettu 16.9.2017).

Kuvia ympäristöstä

Linkki videoon ympäristöstä

https://youtu.be/TkVM4k-k0Jk